COMPUTER VISION - FUNDAMENTALS

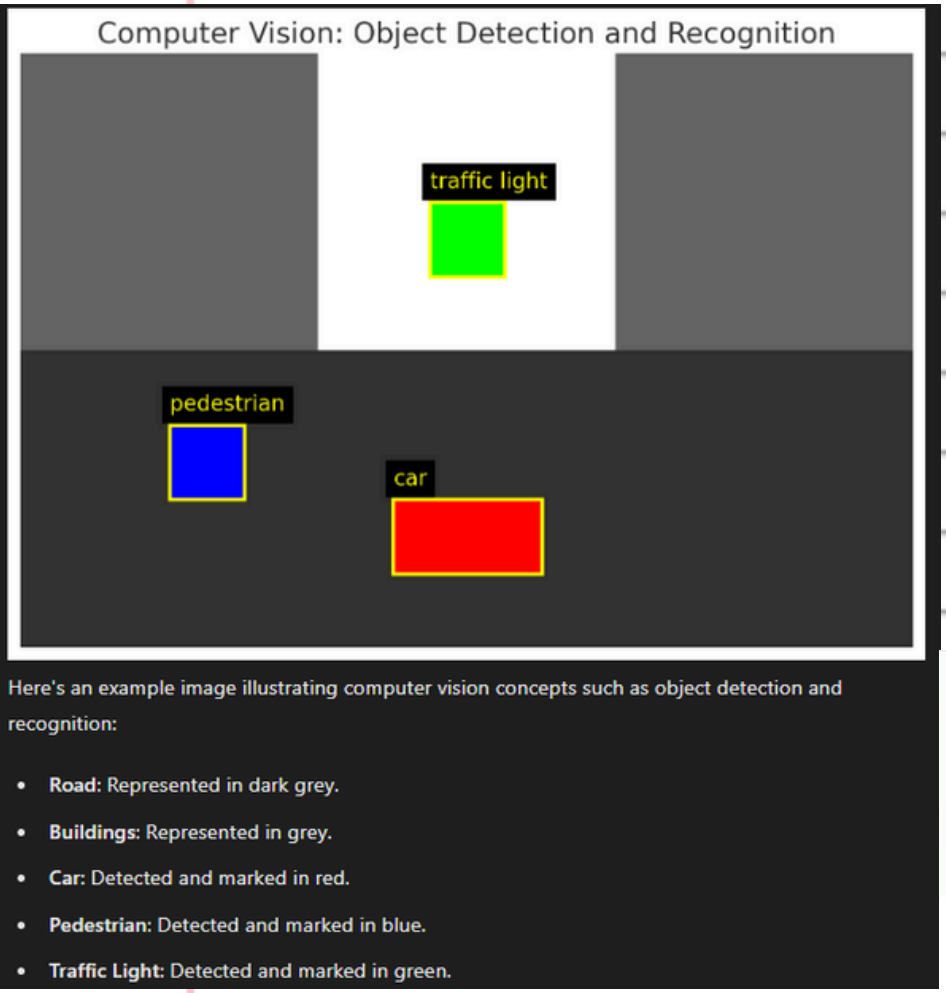
WITH - OPEN CV

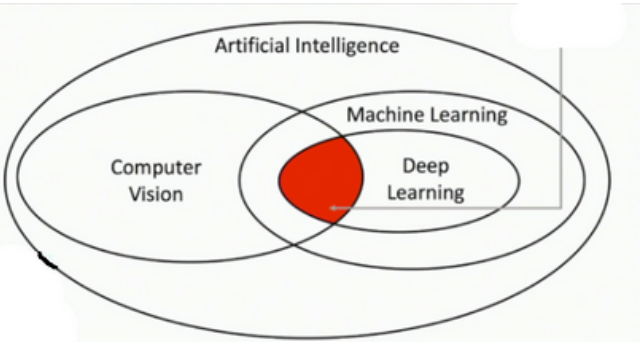
BY - RAJ

THE CONTENT

1. What are Images?

https://docs.opencv.org/4.x/d3/df2/tutorial_py_basic_ops.html





What are images?

- In “most cases” pixel value range from **0** to **255**.
- In binary images pixel value is in **[0, 1]** (or **[0, 255]**).
- In 16 bits images pixel value range from **0** to **65535**.

It is typical for a monochromatic (“black & white”) image to have resolution of 8 bits/pixel. This creates 256 different possible intensity values for each pixel, from black (0) to white (255), with all shades of grey in between. A full-colour image may be quantized to this depth in each of the three colour planes, requiring a total of 24 bits per pixel. However, it is common to represent colour more coarsely or even to combine luminance and chrominance information in such a way that their *total* information is only 8 or 12 bits/pixel.

2 .Image - Input/Output

```python
import cv2

# Read image
image_path = '../Image - Input Output/pic.png'
img = cv2.imread(image_path)


# Write image
output_path = './data/output_pic.png'
cv2.imwrite(output_path, img)

# Visualize image
cv2.imshow( winname: 'window_image_name', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

even can you:
cv2.waitkey(5000)

it will Destroy Window After 5 second

with (0) it means

it means wait until window being closed

```python
import os
import cv2
img = cv2.imread(os.path.join('.', 'dogs.jpg'))
print(img.shape)
cv2.imshow( winname: 'img', img)
cv2.waitKey(0)
```

enough for showing image

Simple 4 Line Code to Display image

```
import cv2                  #import library
img = cv2.imread("../Image.png")    #storing image in valiable

cv2.imshow( winname: "img",img)     #displaying that variable
cv2.waitKey(0)          #wait untill exit key pressed
```

Video - Input/Output

```python
import cv2

# Define video path
video_path = '../Video - Input Output/vlc-record-2024-05-04-22h22m39s-Sna

# Open video file
video = cv2.VideoCapture(video_path)

# Play video
while True:
    ret, frame = video.read()
    if not ret:
        break
    cv2.imshow( winname: 'Video', frame)
    cv2.waitKey(40)

# Release resources
video.release()
cv2.destroyAllWindows()
```

in this release is very important than program will all the memory. and than destroy all the window

these lines were very important

Web Cam - Input/Output

Videocapture: helps to choose the camera its V.Impt

```python
import cv2

# Read webcam
webcam = cv2.VideoCapture(0)

# Visualize webcam
while True:
    frame = webcam.read()[1]

    if frame is None:
        break

    cv2.imshow( winname: 'frame', frame)

    if cv2.waitKey(40) & 0xFF == ord('q'):
        break

webcam.release()
cv2.destroyAllWindows()
```

```python
import cv2

# Read webcam
webcam = cv2.VideoCapture(0)

# Visualize webcam
while True:
    ret, frame = webcam.read()

    cv2.imshow( winname: 'frame', frame)

    if cv2.waitKey(40) & 0xFF == ord('q'):
        break

webcam.release()
cv2.destroyAllWindows()
```

ret - It is a boolean value which is TRUE everytime we successfully read any frame.

3- Basic Operations

resize_image = cv.imread(img, (640,480))

- **Resizing**

image loaded defining size

In This We first get Height and than width

```python
import cv2
# Read the image
img = cv2.imread('../download (1).jpg')

# Resize the image to 640x640
resized_img = cv2.resize(img, dsize: (640, 640))

# Print original and resized image shapes
print("Original image shape:", img.shape)
print("Resized image shape:", resized_img.shape)

# Display the images
cv2.imshow( winname: 'Original Image', img)
cv2.imshow( winname: 'Resized Image', resized_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

cropped_img = img[220:740, 320:940]

- **Croping** defining size by using pixels info.

```python
import cv2
# Read the image
img = cv2.imread('../download (1).jpg')

# Print the original image shape
print("Original image shape:", img.shape)

# Crop the image
cropped_img = img[220:740, 320:940]
#Defining Size     Y-Axix    X_Axis

# Display the images
cv2.imshow( winname: 'Original Image', img)
cv2.imshow( winname: 'Cropped Image', cropped_img)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

"220:740" define y axis and
“320:940" define x axis

4- ColorSpaces

```
img = cv.imread(img)
img_grey = cv.cvtColour(img, cv2.COLOR_BGR2GRAY)
```

Convert Color

Color -BGR to Gray

```python
import cv2

# Read the image
img = cv2.imread('../download (1).jpg')

# Convert the image to different color spaces
img_grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Display the original and HSV images
cv2.imshow( winname: 'Original Image', img)
cv2.imshow( winname: 'Grey Image', img_grey)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

Colorspacing is done to improve the effenciency

5.Bluring

Most Common Use Case Of Blurring Is to remove Noise.

- Smoothing, also called blurring, is a simple and frequently used image processing operation.

https://docs.opencv.org/3.4/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html

Four Functions for Blur:

1. **blur()**
2. **GaussianBlur()**
3. **medianBlur()**
4. **bilateralFilter()**

```python
import cv2

# Read the image

img = cv2.imread('../download (1).jpg')

# Define the kernel size
k_size = 7

# Apply different blurring techniques
img_blur = cv2.blur(img, ksize: (k_size, k_size))
img_gaussian_blur = cv2.GaussianBlur(img, ksize: (k_size, k_size), sigmaX: 5)
img_median_blur = cv2.medianBlur(img, k_size)

# Display the images
cv2.imshow( winname: 'Original Image', img)
cv2.imshow( winname: 'Blurred Image', img_blur)
cv2.imshow( winname: 'Gaussian Blurred Image', img_gaussian_blur)
cv2.imshow( winname: 'Median Blurred Image', img_median_blur)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

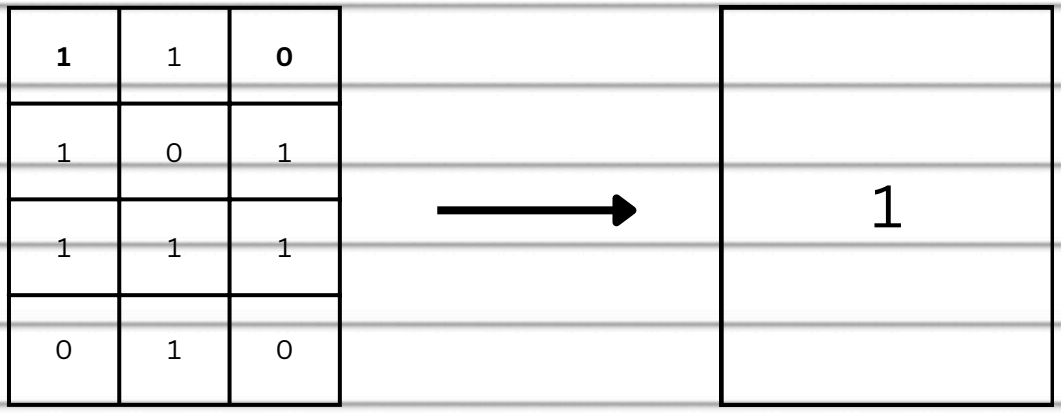
In median Blurring You always take average/mean:

Normalized Box Filter

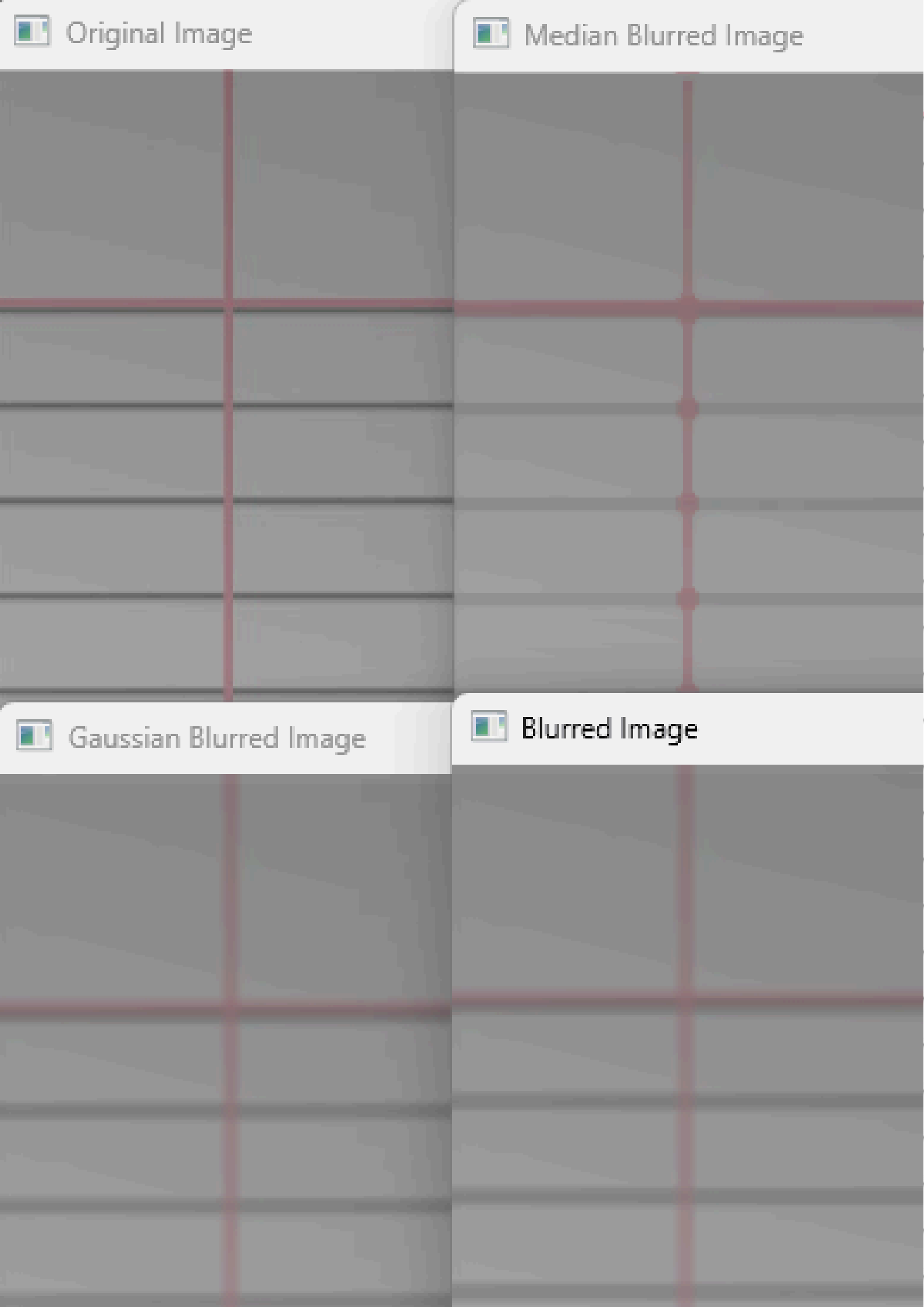
- This filter is the simplest of all! Each output pixel is the *mean* of its kernel neighbors (all of them contribute with equal weights)
- The kernel is below:

$$K = \frac{1}{K_{width} \cdot K_{height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \\ . & . & . & \dots & 1 \\ . & . & . & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

Bade Box of pixels a average or mean will’be alotted to whole pixel




```python
import os

import cv2


img = cv2.imread(os.path.join('.', 'freelancer.jpg'))

k_size = 7
cv2.blur(img, (k_size, k_size))

cv2.imshow('img', img)
cv2.waitKey(0)
```

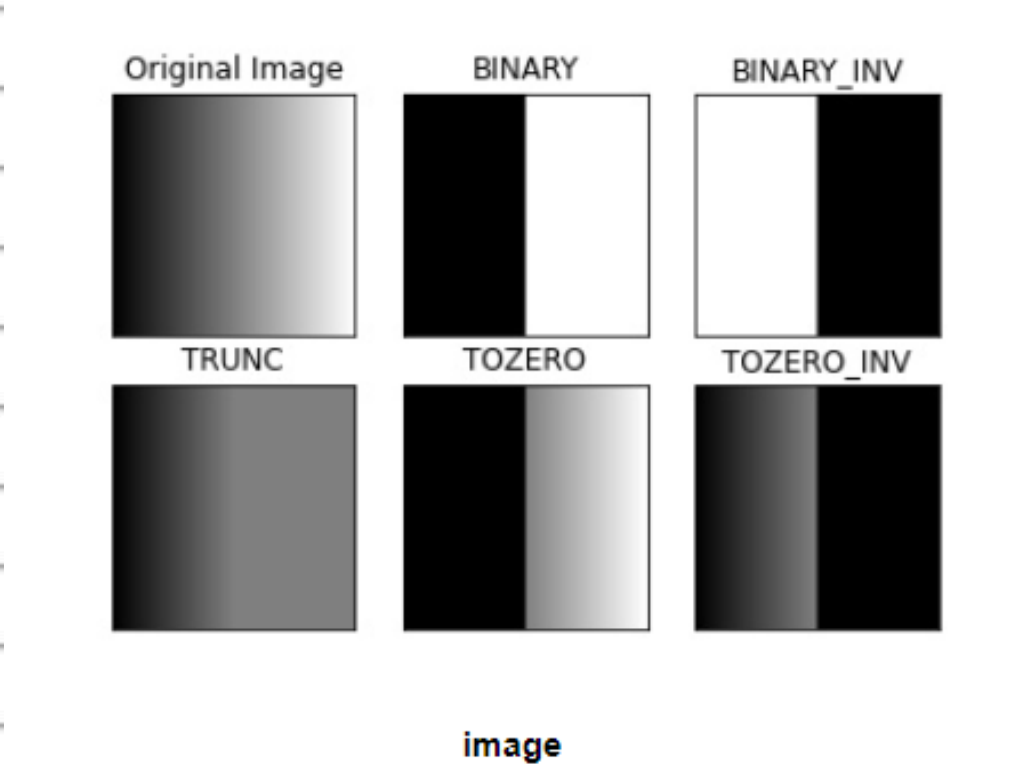

Original Image
Median Blurred Image
Gaussian Blurred Image
Blurred Image

6. Threshold

In this process, a constant threshold value is applied to each pixel in a grayscale image using the cv.threshold function in OpenCV. If a pixel's value is below the threshold, it is set to 0; otherwise, it is set to a specified maximum value. The cv.threshold function takes the source image, the threshold value, the maximum value, and a type parameter which defines the kind of thresholding. The basic type is cv.THRESH_BINARY. Other types include:

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO
- cv.THRESH_TOZERO_INV

Different Types of Threshold:



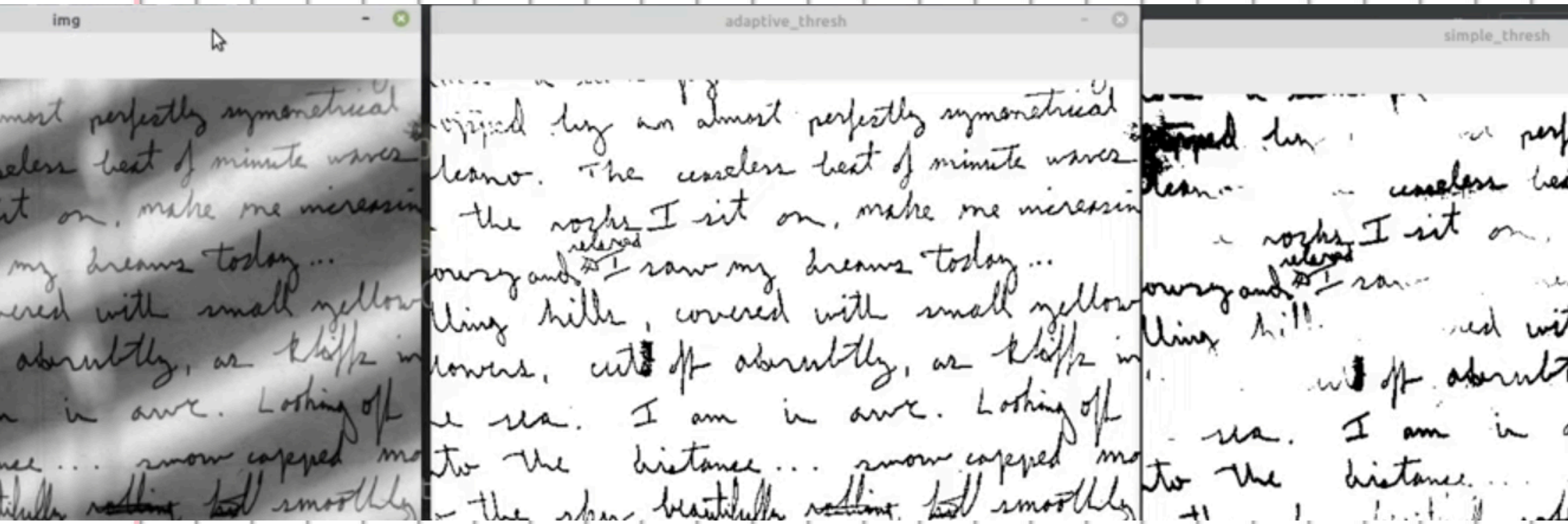
image

- mostly it is used to create a semantic segmentation algorithm right to segment your image into different regions

In This Image Its Easy to remove background Using threshold. Because Bear is Dark in color and grass is light in color.
When You want to perform threshold first you need to convert to gray scale.

https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html#:~:text=Simple%20Thresholding,-Here%2C%20the%20matter&text=The%20function%20cv.threshold%20is,to%20classify%20the%20pixel%20values.

Threshold defination: a level, point, or value above which something is true or will take place and below which it is not or will not


img
adaptive_thresh
simple_thresh

Global Threshold

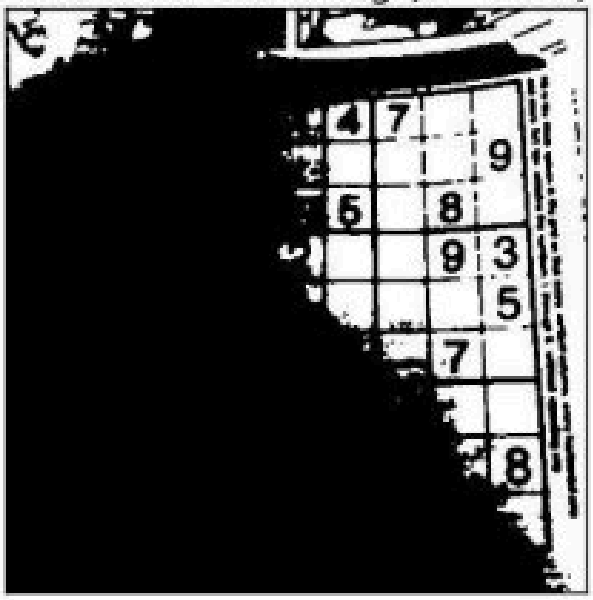
```python
#Global Threshold
import cv2
# Load image
img = cv2.imread('../download (1).jpg')

# Convert to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply thresholding
_, thresh = cv2.threshold(img_gray, thresh: 80, maxval: 255, cv2.THRESH_BINARY)

# Apply blurring
thresh = cv2.blur(thresh, ksize: (10, 10))

# Apply thresholding again
_, thresh = cv2.threshold(thresh, thresh: 80, maxval: 255, cv2.THRESH_BINARY)

# Display the images
cv2.imshow( winname: 'Original Image', img)
cv2.imshow( winname: 'Thresholded Image', thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Adaptive Threshold

```python
#Adaptive Threshold
import cv2
# Load image
img = cv2.imread('../download (1).jpg')

# Convert to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply adaptive thresholding
adaptive_thresh = cv2.adaptiveThreshold(img_gray, maxValue: 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, blockSize: 21, C: 30)

# Apply simple thresholding
_, simple_thresh = cv2.threshold(img_gray, thresh: 80, maxval: 255, cv2.THRESH_BINARY)

# Display the images
cv2.imshow( winname: 'Original Image', img)
cv2.imshow( winname: 'Adaptive Threshold', adaptive_thresh)
cv2.imshow( winname: 'Simple Threshold', simple_thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Original Image

Global Thresholding (v = 127)



Adaptive Mean Thresholding

Adaptive Gaussian Thresholding

7. Edge Detection

Edge detection in computer vision identifies sharp changes in image intensity, marking object boundaries. Common methods include the Sobel, Prewitt, and Canny detectors. Steps involve noise reduction, gradient calculation, non-maximum suppression, and edge tracking to enhance image analysis and object recognition.

Steps in Edge Detection

1. **Noise Reduction**:
   - Images often contain noise that can affect edge detection. Applying a smoothing filter, such as a Gaussian blur, helps to reduce noise and avoid false edge detection.
2. **Gradient Calculation**:
   - The gradient of the image is computed to detect regions with high spatial derivatives. The gradient indicates the direction and rate of the most rapid change in intensity.
3. **Edge Orientation**:
   - The direction of the gradient is used to determine the orientation of the edge. This is important for later steps like non-maximum suppression.
4. **Non-Maximum Suppression**:
   - This step involves thinning out the edges by suppressing all the gradient values except the local maxima. This means only the points with the highest gradient in the direction of the gradient are retained.
5. **Double Thresholding**:
   - Two threshold values are applied to categorize edges into strong, weak, and non-edges. Strong edges are definitely edges, weak edges are potential edges, and non-edges are discarded.

6. **Edge Tracking by Hysteresis:**
    - Weak edges that are connected to strong edges are retained, while those that are not connected are discarded. This helps in preserving the continuity of edges.

Common Edge Detection Algorithms

1. Sobel Operator

- The Sobel operator uses convolutional kernels to approximate the gradient of the image intensity function. It uses two 3x3 kernels, one for detecting horizontal edges and one for vertical edges. The result is a gradient magnitude and direction.

2. Prewitt Operator

- Similar to the Sobel operator, the Prewitt operator also uses convolutional kernels but with slightly different weights. It is less sensitive to noise compared to the Sobel operator.

3. Canny Edge Detector

- Developed by John F. Canny, this is one of the most widely used edge detection algorithms due to its effectiveness and accuracy. It includes all the steps mentioned above: noise reduction, gradient calculation, non-maximum suppression, double thresholding, and edge tracking by hysteresis.

4. Laplacian of Gaussian (LoG)

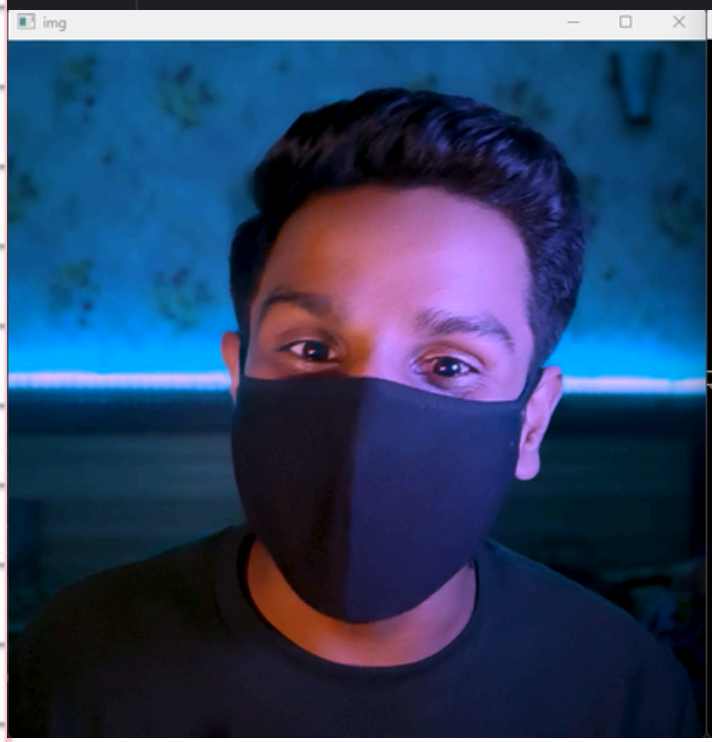
- This method involves applying a Gaussian filter to smooth the image, followed by the Laplacian filter to detect edges. The zero-crossings of the Laplacian are used to locate the edges.

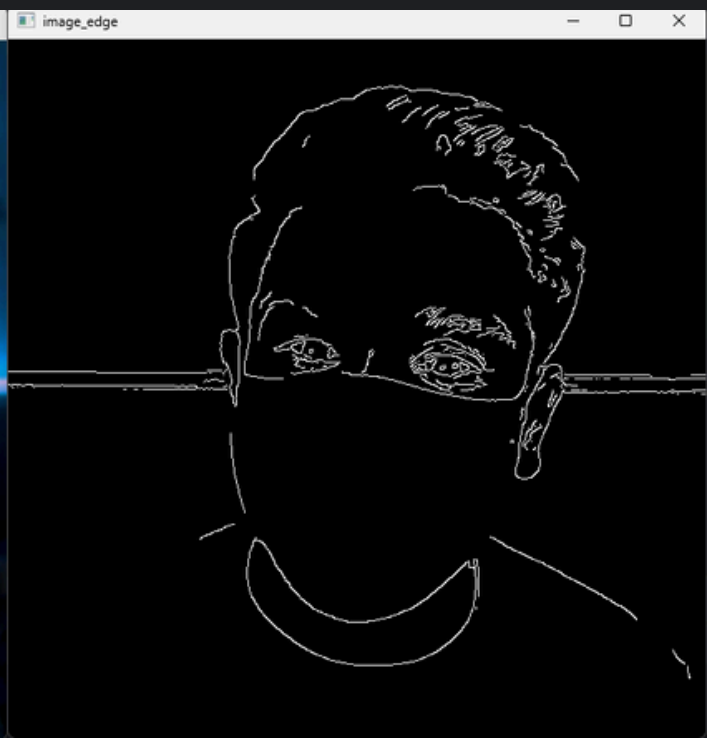
5. Roberts Cross Operator

- The Roberts Cross operator uses a pair of 2x2 convolutional kernels to approximate the gradient. It is simple and fast but more sensitive to noise.

Example: Using the Canny Edge Detector in OpenCV

```python
import cv2                  #import library
img = cv2.imread("../Image.png")    #storing image in valiable

image_edge = cv2.Canny(img, 100, 200)

cv2.imshow( winname: "image_edge",image_edge)
cv2.imshow( winname: "img",img)     #displaying that variable
cv2.waitKey(0)      #wait untill exit key pressed

```

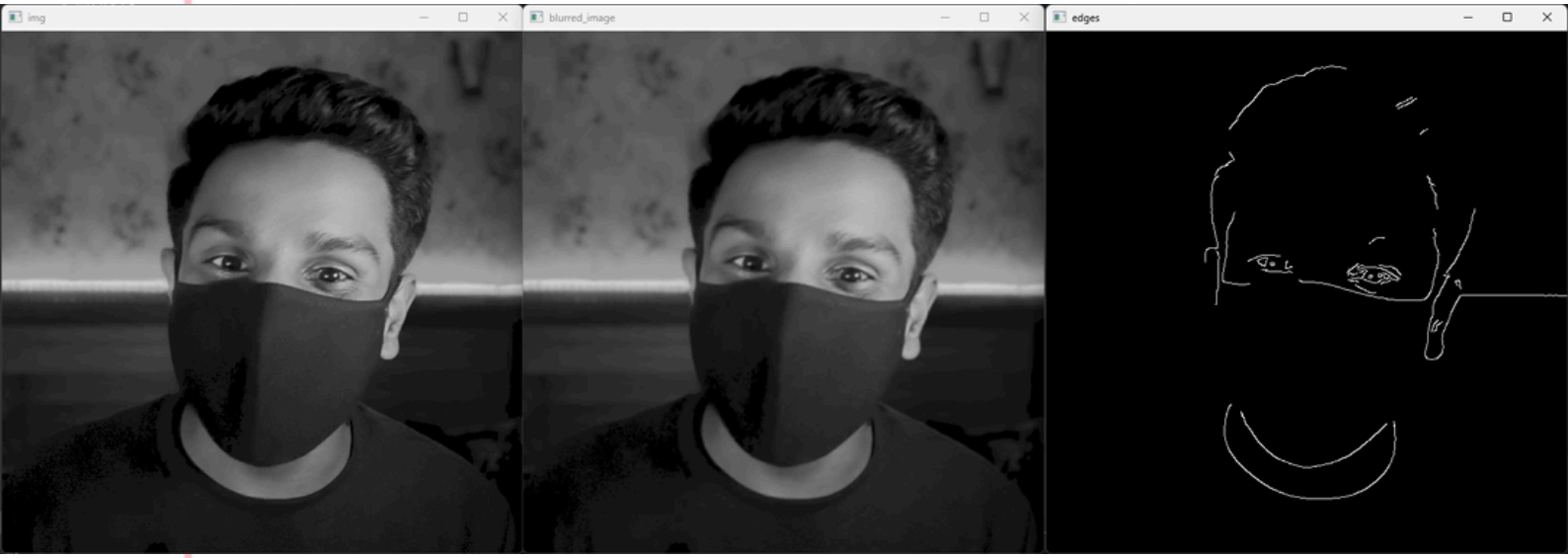





```
Canny(image, threshold1, threshold2, edges, apertureSize, L2gradient)        cv2
```

After Applying GRAYSCALE, GaussianBlur & Canny

```python
import cv2                #import library
img = cv2.imread("../Image.png",cv2.IMREAD_GRAYSCALE)    #storing image in valiable


# Apply Gaussian blur to reduce noise
blurred_image = cv2.GaussianBlur(img, ksize: (3, 3), sigmaX: 1.4)

# Apply Canny edge detector
edges = cv2.Canny(blurred_image, 100, 200)


cv2.imshow( winname: "blurred_image",blurred_image)
cv2.imshow( winname: "edges",edges)
cv2.imshow( winname: "img",img)     #displaying that variable
cv2.waitKey(0)       #wait untill exit key pressed
```



Theory

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in

1. It is a multi-stage algorithm and we will go through each stages.
2. **Noise Reduction**

   Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

3. **Finding Intensity Gradient of the Image**

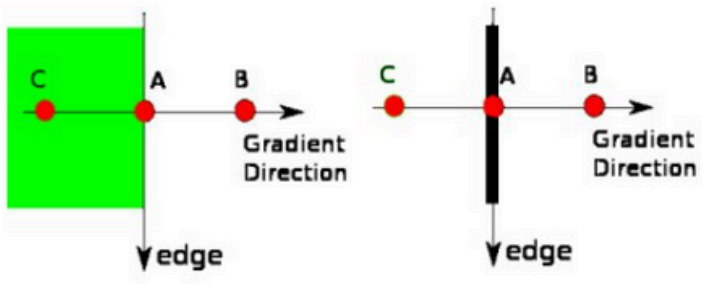
   Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2} Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

1. **Non-maximum Suppression**

   After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:



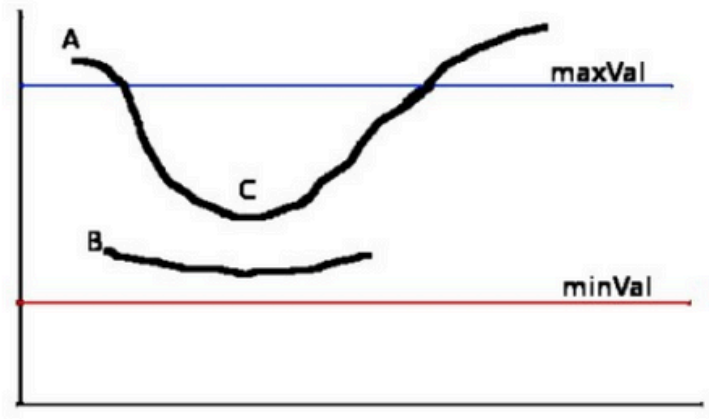
image

Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

In short, the result you get is a binary image with "thin edges".

1. **Hysteresis Thresholding**

   This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:



image

The edge A is above the maxVal, so considered as "sure-edge". Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.

This stage also removes small pixels noises on the assumption that edges are long lines.

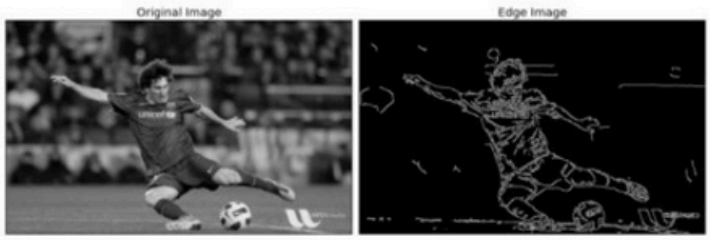
So what we finally get is strong edges in the image.

Canny Edge Detection in OpenCV

OpenCV puts all the above in single function, **cv.Canny()**. We will see how to use it. First argument is our input image. Second and third arguments are our minVal and maxVal respectively. Fourth argument is aperture_size. It is the size of Sobel kernel used for find image gradients. By default it is 3. Last argument is L2gradient which specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned above which is more accurate, otherwise it uses this function: $Edge\_Gradient\ (G) = |G_x| + |G_y|$. By default, it is False.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('messi5.jpg', cv.IMREAD_GRAYSCALE)
assert img is not None, "file could not be read, check with os.path.exists()"
edges = cv.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])

plt.show()
```

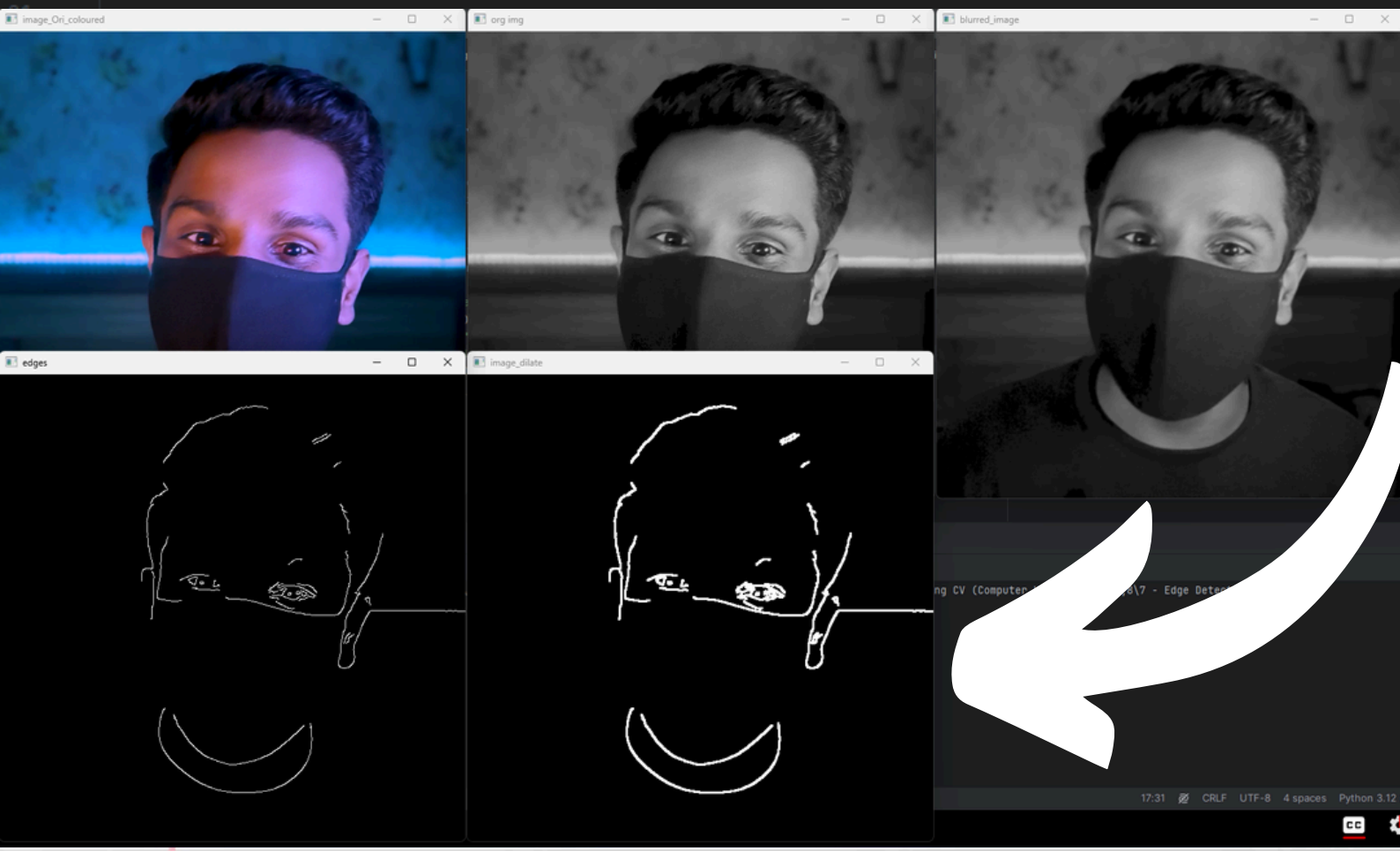
See the result below:



image

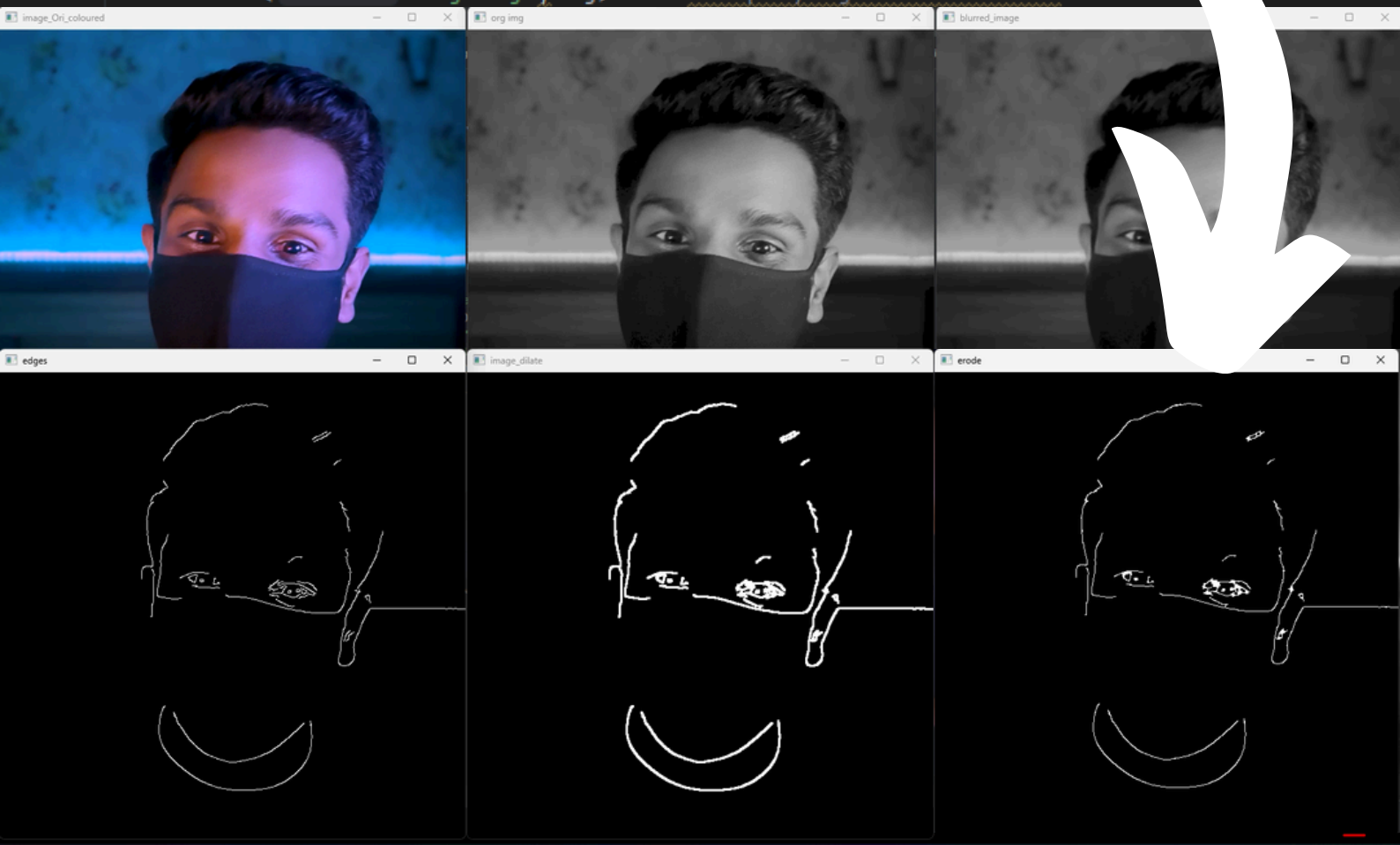
Using Dilate : Used to make Edge Thicker

```python
import cv2
import numpy as np
imgo = cv2.imread("../Image.png")
img = cv2.imread("../Image.png",cv2.IMREAD_GRAYSCALE)    #storing image in valiable


# Apply Gaussian blur to reduce noise
blurred_image = cv2.GaussianBlur(img, ksize: (3, 3), sigmaX: 1.4)

# Apply Canny edge detector
edges = cv2.Canny(blurred_image, 100, 200)
image_del = cv2.dilate(edges,np.ones( shape: (3, 3), dtype=np.int8)) #Dilate

cv2.imshow( winname: "blurred_image",blurred_image)
cv2.imshow( winname: "edges",edges)
cv2.imshow( winname: "image_dilate",image_del)
cv2.imshow( winname: "image_Ori_coloured",imgo)
cv2.imshow( winname: "org img",img)     #displaying that variable
cv2.waitKey(0)      #wait untill exit key pressed

```



Using Erosion : Border Much thicker like after using Canny only Edge

```python
import cv2
import numpy as np
imgo = cv2.imread("../Image.png")
img = cv2.imread("../Image.png",cv2.IMREAD_GRAYSCALE)    #storing image in valiable


# Apply Gaussian blur to reduce noise
blurred_image = cv2.GaussianBlur(img, ksize: (3, 3), sigmaX: 1.4)

# Apply Canny edge detector
edges = cv2.Canny(blurred_image, 100, 200)
image_del = cv2.dilate(edges,np.ones( shape: (3, 3), dtype=np.int8)) #Dilate
img_edge_e = cv2.erode(image_del, np.ones( shape: (3, 3), dtype=np.int8))

cv2.imshow( winname: "blurred_image",blurred_image)
cv2.imshow( winname: "edges",edges)
cv2.imshow( winname: "image_dilate",image_del)
cv2.imshow( winname: "image_Ori_coloured",imgo)
cv2.imshow( winname: "erode",img_edge_e)
cv2.imshow( winname: "org img",img)     #displaying that variable
```



1. Importing Libraries

```python
import cv2
import numpy as np
```

- `cv2`: This is the OpenCV library, which is used for image processing tasks.
- `numpy`: This is the NumPy library, which is used for numerical operations and array manipulation.

2. Reading the Image

```python
imgo = cv2.imread("../Image.png")
img = cv2.imread("../Image.png", cv2.IMREAD_GRAYSCALE)
```

- `cv2.imread("../Image.png")`: This function reads the image file named 'Image.png' from the specified path and loads it into the variable `imgo`. This is the original image in color.
- `cv2.imread("../Image.png", cv2.IMREAD_GRAYSCALE)`: This function reads the same image but converts it to grayscale. The grayscale image is stored in the variable `img`.

3. Applying Gaussian Blur

```python
blurred_image = cv2.GaussianBlur(img, (3, 3), 1.4)
```

- `cv2.GaussianBlur(img, (3, 3), 1.4)`: This function applies a Gaussian blur to the grayscale image `img` to reduce noise and detail.
  - `(3, 3)`: This specifies the size of the kernel used for blurring.
  - `1.4`: This is the standard deviation of the Gaussian kernel.

4. Applying Canny Edge Detector

```python
edges = cv2.Canny(blurred_image, 100, 200)
```

- `cv2.Canny(blurred_image, 100, 200)`: This function applies the Canny edge detection algorithm to the blurred image `blurred_image`.
  - `100`: This is the lower threshold for the hysteresis procedure.
  - `200`: This is the upper threshold for the hysteresis procedure.
  - The result is stored in the variable `edges`, which contains the detected edges.

5. Dilating the Edges

```python
image_del = cv2.dilate(edges, np.ones((3, 3), dtype=np.int8))
```

- `cv2.dilate(edges, np.ones((3, 3), dtype=np.int8))`: This function dilates the edges in the `edges` image.
  - `np.ones((3, 3), dtype=np.int8)`: This creates a 3x3 kernel of ones, which is used for dilation. Dilation increases the thickness of the edges.

6. Eroding the Dilated Image

```python
img_edge_e = cv2.erode(image_del, np.ones((3, 3), dtype=np.int8))
```

- `cv2.erode(image_del, np.ones((3, 3), dtype=np.int8))`: This function erodes the dilated image `image_del`.
  - `np.ones((3, 3), dtype=np.int8)`: This creates a 3x3 kernel of ones, which is used for erosion. Erosion reduces the thickness of the edges.

7. Displaying the Images

```python
cv2.imshow("blurred_image", blurred_image)
cv2.imshow("edges", edges)
cv2.imshow("image_dilate", image_del)
cv2.imshow("image_Ori_coloured", imgo)
cv2.imshow("erode", img_edge_e)
cv2.imshow("org img", img)
cv2.waitKey(0)
```

8. Drawing

Define : cv2 drawing functions in OpenCV allow you to create shapes, lines, and text on images using functions like `cv2.line`, `cv2.rectangle`, `cv2.circle`, and `cv2.putText`, among others.

Goal

- Learn to draw different geometric shapes with OpenCV
- You will learn these functions : **cv.line()**, **cv.circle()** , **cv.rectangle()**, **cv.ellipse()**, **cv.putText()** etc.

Code

In all the above functions, you will see some common arguments as given below:

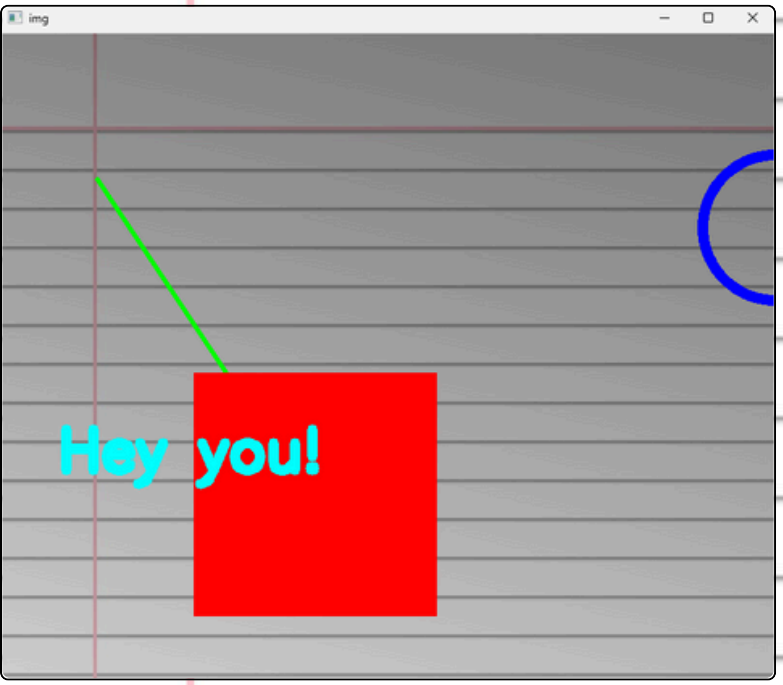
- img : The image where you want to draw the shapes
- color : Color of the shape. for BGR, pass it as a tuple, eg: (255,0,0) for blue. For grayscale, just pass the scalar value.
- thickness : Thickness of the line or circle etc. If **-1** is passed for closed figures like circles, it will fill the shape. *default thickness = 1*
- lineType : Type of line, whether 8-connected, anti-aliased line etc. *By default, it is 8-connected.* **cv.LINE_AA** gives anti-aliased line which looks great for curves.

```python
import cv2

# Load the image
img = cv2.imread('../download (1).jpg')

# Print image dimensions
print(img.shape)

# Draw a green line
cv2.line(img, pt1: (100, 150), pt2: (300, 450), color: (0, 255, 0), thickness: 3)

# Draw a filled red rectangle
cv2.rectangle(img, (200, 350), (450, 600), (0, 0, 255), -1)

# Draw a blue circle with a thick border
cv2.circle(img, center: (800, 200), radius: 75, color: (255, 0, 0), thickness: 10)

# Add yellow text
cv2.putText(img, text: 'Hey you!', org: (600, 450), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 2, color: (255, 255, 0), thickness: 10)

# Display the image
cv2.imshow( winname: 'img', img)
cv2.waitKey(0)
```

Explaining With Line Draw:

cv2.line(img, (100, 150), (300, 450), (0, 255, 0), 3)

- This is the image on which the line will be drawn. It is assumed to be previously loaded
- This is the starting point of the line. It is a tuple representing the (x, y) coordinates of the starting point
- This is the ending point of the line. It is a tuple representing the (x, y) coordinates of the ending point.
- This is the color of the line in BGR (Blue, Green, Red) format. In this case, (0, 255, 0) represents green.
- This is the thickness of the line in pixels.



- `cv2.line`: This is the OpenCV function used to draw a line on an image.
- `img`: This is the image on which the line will be drawn. It is assumed to be previously loaded using `cv2.imread`.
- `(100, 150)`: This is the starting point of the line. It is a tuple representing the (x, y) coordinates of the starting point.
- `(300, 450)`: This is the ending point of the line. It is a tuple representing the (x, y) coordinates of the ending point.
- `(0, 255, 0)`: This is the color of the line in BGR (Blue, Green, Red) format. In this case, (0, 255, 0) represents green.
- `3`: This is the thickness of the line in pixels.

Explaining With Adding Text:

```
cv2.putText(img, 'Hey you!', (60, 450), cv2.FONT_HERSHEY_SIMPLEX, 2,
(255, 255, 0), 10)
```

This is the image on which the line will be drawn. It is assumed to be previously loaded

This specifies the font type. 'cv2.FONT_HERSHEY_SIMPLEX' is a normal sans-serif font.

```
text: 'Hey you!',

fontScale: 2, color: (255, 255, 0), thickness: 10)
```

- `cv2.putText`: This is the function from the OpenCV library that puts text on an image.
- `img`: This is the image on which the text will be drawn.
- `'Hey you!'`: This is the text string that will be added to the image.
- `(600, 450)`: This is the position where the text will start on the image. It's given as (x, y) coordinates.
- `cv2.FONT_HERSHEY_SIMPLEX`: This specifies the font type. `cv2.FONT_HERSHEY_SIMPLEX` is a normal sans-serif font.
- `2`: This is the font scale factor that is multiplied by the font-specific base size. It makes the text larger or smaller.
- `(255, 255, 0)`: This is the color of the text in BGR format. (255, 255, 0) represents the color yellow.
- `10`: This is the thickness of the lines used to draw the text. Higher values make the text thicker.

9. Contours

Define: OpenCV cv2.findContours detects object boundaries in images, returning contours as point lists. Useful for shape analysis and object detection, typically post-thresholding or edge detection.

What are contours?

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

- For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
- Since OpenCV 3.2, **findContours()** no longer modifies the source image but returns a modified image as the first of three return parameters.
- In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

Contour Approximation Method

This is the third argument in **cv.findContours** function. What does it denote actually?

Above, we told that contours are the boundaries of a shape with same intensity. It stores the (x,y) coordinates of the boundary of a shape. But does it store all the coordinates ? That is specified by this contour approximation method.

If you pass **cv.CHAIN_APPROX_NONE**, all the boundary points are stored. But actually do we need all the points? For eg, you found the contour of a straight line. Do you need all the points on the line to represent that line? No, we need just two end points of that line. This is what **cv.CHAIN_APPROX_SIMPLE** does. It removes all redundant points and compresses the contour, thereby saving memory.

Below image of a rectangle demonstrate this technique. Just draw a circle on all the coordinates in the contour array (drawn in blue color). First image shows points I got with **cv.CHAIN_APPROX_NONE** (734 points) and second image shows the one with **cv.CHAIN_APPROX_SIMPLE** (only 4 points). See, how much memory it saves!!!

https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html

```python
import cv2

# Read the image
img = cv2.imread('../birds.jpg')
original_img1 = cv2.imread('../birds.jpg')

# Convert the image to grayscale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply a binary threshold to the grayscale image
_, thresh = cv2.threshold(img_gray, thresh: 127, maxval: 255, cv2.THRESH_BINARY_INV)

# Find contours in the thresholded image
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

# Draw rectangles around contours with an area greater than 200
for cnt in contours:
    if cv2.contourArea(cnt) > 200:
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)

# Display the original image with rectangles and the thresholded image
cv2.imshow( winname: 'Image', img)
cv2.imshow( winname: 'original_img1', original_img1)
cv2.imshow( winname: 'Threshold', thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

1. Importing Libraries

python Copy code

```python
import cv2
```

- `cv2`: This is the OpenCV library, which is used for image processing tasks.

2. Reading the Image

python Copy code

```python
img = cv2.imread('birds.jpg')
```

- `cv2.imread('birds.jpg')`: This function reads the image file named 'birds.jpg' and loads it into the variable `img`.

3. Converting the Image to Grayscale

```python
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

- `cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)`: This function converts the color image `img` from BGR (Blue, Green, Red) color space to grayscale. The resulting grayscale image is stored in `img_gray`.

4. Applying a Binary Threshold

```python
_, thresh = cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY_INV)
```

- `cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY_INV)`: This function applies a binary threshold to the grayscale image `img_gray`.
  - If a pixel value is greater than 127, it is set to 0 (black).
  - If a pixel value is less than or equal to 127, it is set to 255 (white).
  - The result is an inverted binary image (`cv2.THRESH_BINARY_INV`), where the object is white on a black background. The result is stored in `thresh`.

5. Finding Contours

```python
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

- `cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)`: This function finds the contours in the binary image `thresh`.
  - `cv2.RETR_TREE`: This mode retrieves all of the contours and reconstructs a full hierarchy of nested contours.
  - `cv2.CHAIN_APPROX_SIMPLE`: This method compresses horizontal, vertical, and diagonal segments and leaves only their end points.
  - The resulting contours are stored in `contours`.

6. Drawing Rectangles Around Contours

```python
for cnt in contours:
    if cv2.contourArea(cnt) > 200:
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

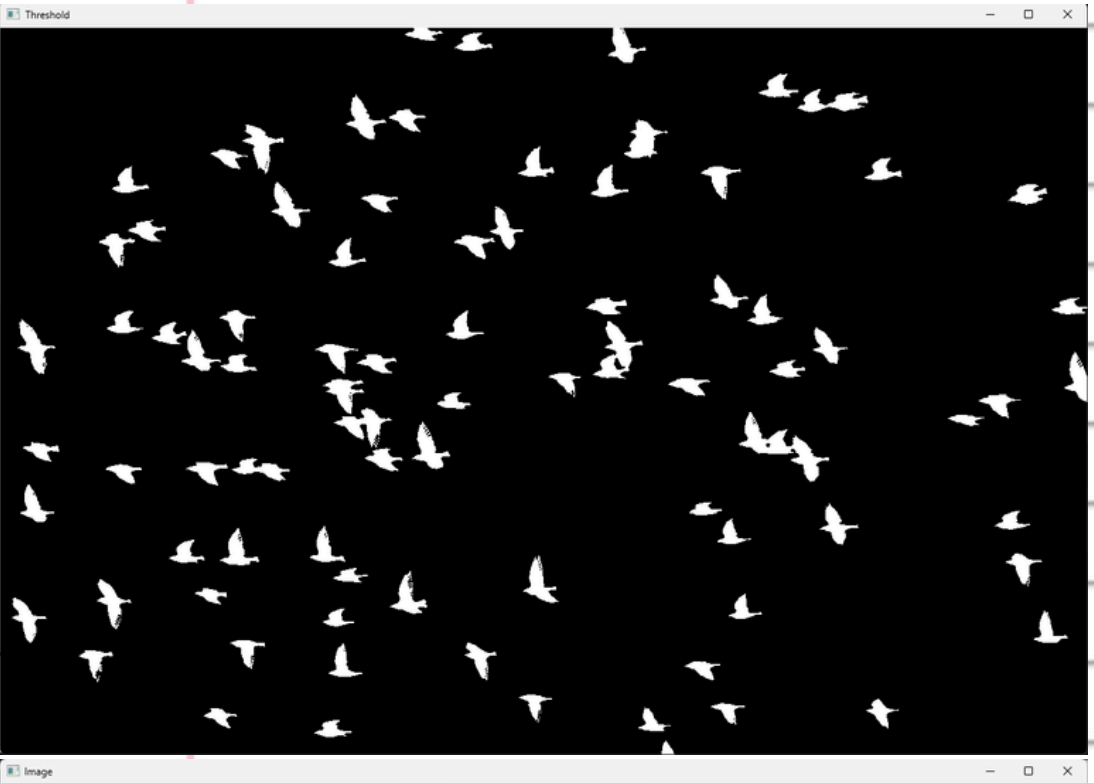
- This loop iterates through each contour in `contours`.
  - `cv2.contourArea(cnt)`: This function calculates the area of the contour `cnt`. The if-statement ensures that only contours with an area greater than 200 pixels are processed.
  - `cv2.boundingRect(cnt)`: This function calculates the bounding rectangle for the contour `cnt`, returning the x and y coordinates of the top-left corner and the width and height of the rectangle.
  - `cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 2)`: This function draws a green rectangle around the contour on the original image `img`.

7. Displaying the Images

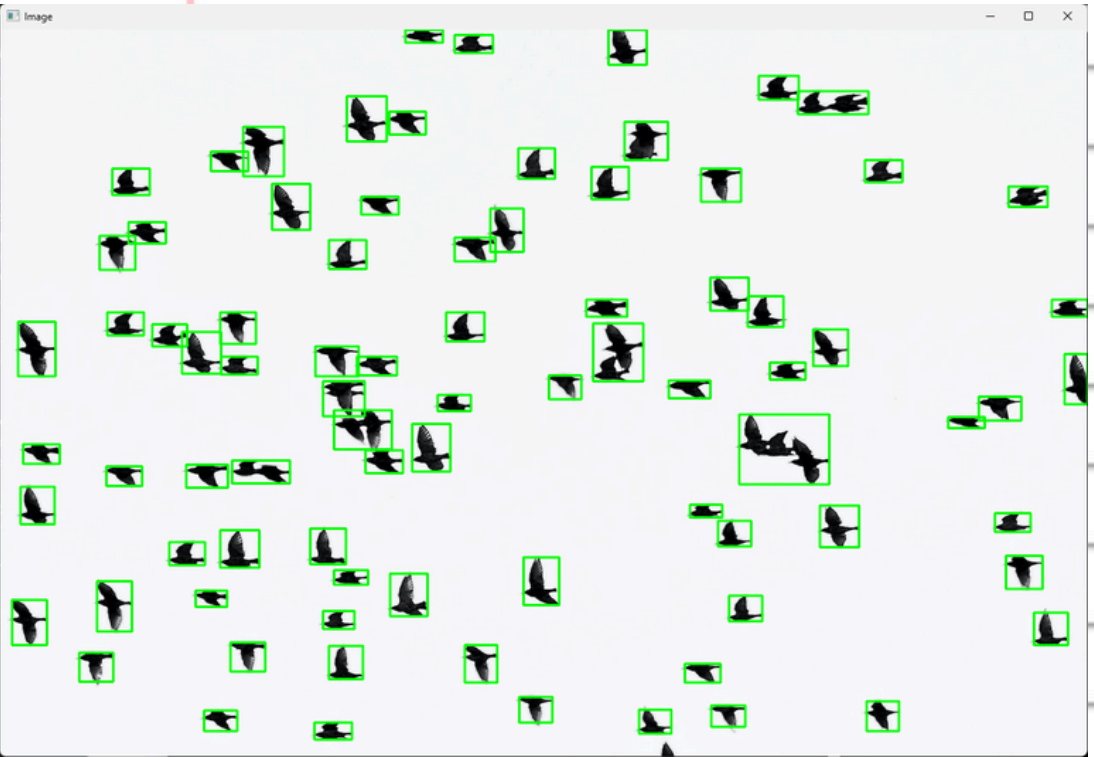
```python
cv2.imshow('Image', img)
cv2.imshow('Threshold', thresh)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- `cv2.imshow('Image', img)`: This function displays the original image with rectangles drawn around the contours in a window titled 'Image'.
- `cv2.imshow('Threshold', thresh)`: This function displays the binary thresholded image in a window titled 'Threshold'.

original Image



Threshold Inverse Image



Contours Image